PASSED

Computational Statistics Lab 1

Niclas Lovsjö, Maxime Bonneau

2 februari 2016

Assignment 1: "Be careful with '=='"

a)

Using the following code we will see how R handles the decimals of fractions,

```
x1<-1/3;
x2<-1/4;
if ((x1-x2)==1/12){
print("Teacher said true")
} else{
print("Teacher lied")}
```

```
## [1] "Teacher lied"
```

Why is this? Looking at the decimals that R stores in x1 gives,

```
print(x1,digits=22)
```

```
## [1] 0.3333333333333333148296
```

which is obviously different from the real number $1/3 = 0.3333....$ This is the reason for 'Teacher lied', but why does R save x1 like this? x1 is an approximation of 1/3 such that it fits the number of bits available.

b)

How can we modify the program to give the correct answer? We know that the amount of floating-points numbers are the same in the interval $[b^i, b^{i+1}]$ as in the interval $[b^{i+1}, b^{i+2}]$. For the base $b = 2$ this yields that the latter interval is twice the size of the former, but they have the same amount of floating-points. In other words, if we have some number $a \in \mathbb{R}$ close to zero then there will be more possible approximations of a then if a is further away from zero. To see this, let us run the program again, and simply add the integer 1 to both sides of the boolean to get further away from zero:

```
x1<-1/3;
x2<-1/4;
if (1+(x1-x2)==1+1/12){
print("Teacher said true")
} else{
print("Teacher lied")}
```

```
## [1] "Teacher said true"
```

Another manipulation would be to move over x2 to the right hand side to keep both sides larger, which would also yield "Teacher said true".

Assignment 2: "Derivative"

a)

We will use the definition of the derivative to find the value of $f'(x)$ for $f(x) = x$.

b)

We compute the derivative for $x = 100000$:

```
f_prime(100000,10^{-15})
```

```
## [1] 0
```

c)

The derivative of $f(x) = x$ is $f'(x) = 1$, so why do we get the result 0? When trying to add a big number with a really small number in R, there are not enough bits to represent the small ϵ-part at the same time as the representation of 100000. What it does is this:

```
print(10^{-15},digits=22)
```

```
## [1] 1.000000000000000077705e-15
```

```
print(100000+10^{-15},digits=22)
```

```
## [1] 1e+05
```

So we see that it actually counts $\frac{f(x+\epsilon)-f(x)}{\epsilon} = \frac{100000-100000}{10^{-15}} = 0$.

Assignment 3: "Variance"

a)

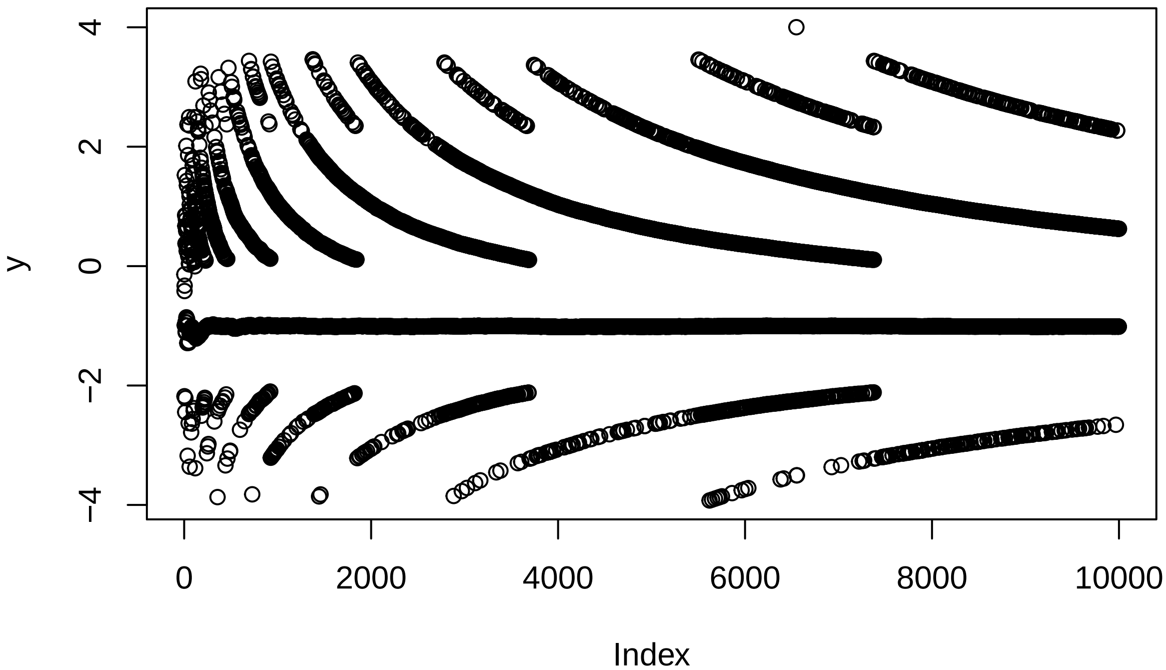
The variance of a R.V. X can be written, $var(X) = \frac{1}{n-1}\left(\sum X_i^2 - \frac{1}{n}\left(\sum X_i\right)^2\right)$. We create an R-function that does this computation.

b)

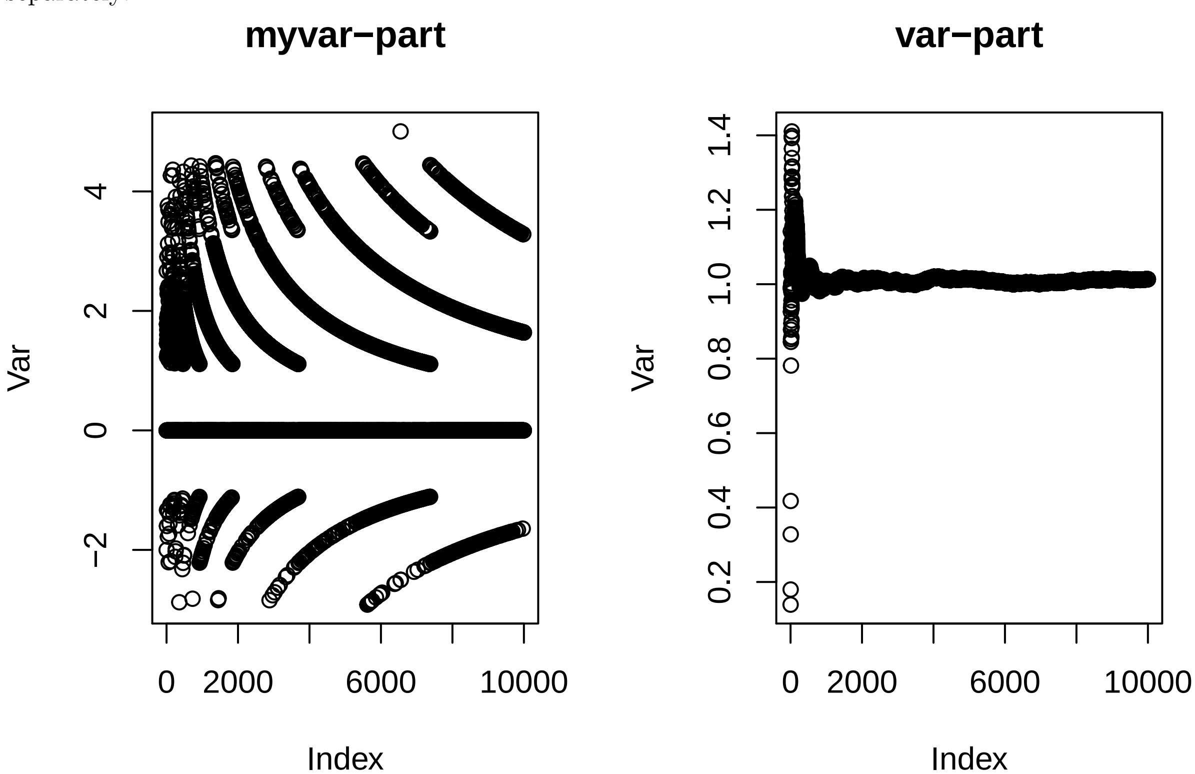
Let us generate random numbers $X_i, i = 1, ..., 10000$ with $X_i \sim N(10^8, 1)$.

c)

Then we calculate the difference $Y_j = myvar(x) - var(x)$ for each subset $X_j = (X_1, ..., X_j)$, $j \in [1, ..., 10000]$ and plot the dependence of Y_j on j.



What is the reason for this strange appearance? We can have a look at the $myvar(x)$ and $var(x)$-part of Y_j separately:



So we see that this behaviour only comes from myvar(). We think this is due to that when the numbers are that large as $\sim 10^{2*8}$ there are not enough distinct number representations when approximating the real with some float. This means that the reals will be given floating values in clusters, such that alot of the reals will have the same float for both the $\sum X^2$-part and the $\frac{1}{n}(\sum X)^2$-part. Others will be assigned some fixed number(one of the existing floats), which then will create a pattern for myvar().

Assignment 4: Linear algebra

a,b,c)

We will use the known data tecator and experiment with solving linear equations using the function solve(). We read in the data and try to solve the equation $A\beta = b$ by solve(A,b). This gives the following error, "Error in solve.default(A, b) : system is computationally singular: reciprocal condition number = 3.02468e-17".

This is due to the fact that the values are really close from each other in several columns, which are therefore considered as linearly dependent, and so A cannot be inverted.

d)

The problem stays when we try to compute the condition number, since we need to invert A, using the definition of condition numbers. We could compute the condition number using the eigenvalue-method since A is normal.

e)

After scaling, the error does not appear anymore. We now get a solution because the values of X and Y are scaled, so around 1, the difference between the columns is more significant, hence the computer does not see the feature-vectors as linearly dependant now.

The solution, i.e. the β-coefficients looks like this for the first 5 values:

```
## [1] -4.959026e-03  7.252746e+01 -5.015684e+02  5.000584e+02 -4.747686e+02
## [6]  5.861003e+02
```

and the condition number for the scaled variables is,

```
## The condition number is: 5.29523e+14
```

Code:

```
library(knitr)
opts_chunk$set(echo=FALSE)
setwd("/Users/niclaslovsjo/Library/Mobile Documents/com~apple~CloudDocs/Kurser/Comp stat/t1")
x1<-1/3;
x2<-1/4;
if ((x1-x2)==1/12){
print("Teacher said true")
} else{
print("Teacher lied")}
print(x1,digits=22)
x1<-1/3;
x2<-1/4;
if (1+(x1-x2)==1+1/12){
print("Teacher said true")
} else{
print("Teacher lied")}
f_prime<-function(x,eps){
  f<-function(x){return(x)}
  result<-(f(x+eps)-f(x))/eps
  return(result)
}
f_prime(100000,10^{-15})
print(10^{-15},digits=22)
print(100000+10^{-15},digits=22)
myvar<-function(x){
  res<-1/(length(x)-1)*(sum(x^2)-1/length(x)*(sum(x))^2)
  return(res)
}
x<-rnorm(10000,10^8,1)
y<-c()
for (i in 1:10000){
  y[i]<-myvar(x[1:i])-var(x[1:i])
}
plot(y)
ex_myvar<-c()
for (i in 1:10000){
  ex_myvar[i]<-myvar(x[1:i])
}
ex_var<-c()
for (i in 1:10000){
  ex_var[i]<-var(x[1:i])
}
par(mfrow=c(1,2))
plot(ex_myvar,main="myvar-part",ylab="Var")
plot(ex_var,main="var-part",ylab="Var")


# 4.
library(xlsx)
data <- read.xlsx("/Users/niclaslovsjo/Library/Mobile Documents/com~apple~CloudDocs/Kurser/Comp stat/t1/
X <- as.matrix(data[,-(ncol(data)-1)])
Y <- as.matrix(data[,ncol(data)-1])
```

```r
A <- t(X)%*%X
b <- t(X)%*%Y
#solve(A, b)
#cn <- norm(A)*norm(solve(A))

# 5.
X2 <- scale(X,scale = TRUE,center=FALSE)
Y2 <- scale(Y,scale = TRUE,center=FALSE)
A2 <- t(X2)%*%X2
b2 <- t(X2)%*%Y2
head(as.vector(solve(A2, b2)))
cn <- norm(A2)*norm(solve(A2))
cat("The condition number is:",cn)
## NA
```